

Отчёт по лабораторной работе №12

**Программирование в командном процессоре ОС UNIX. Расширенное
программирование**

Щербак Маргарита Романовна

2022

1 Цель работы:

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

1.1 Задание:

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (`> /dev/tty#`, где `#` — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.
2. Реализовать команду `man` с помощью командного файла. Изучите содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое справки. Командный файл должен получать в

виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге `man1`.

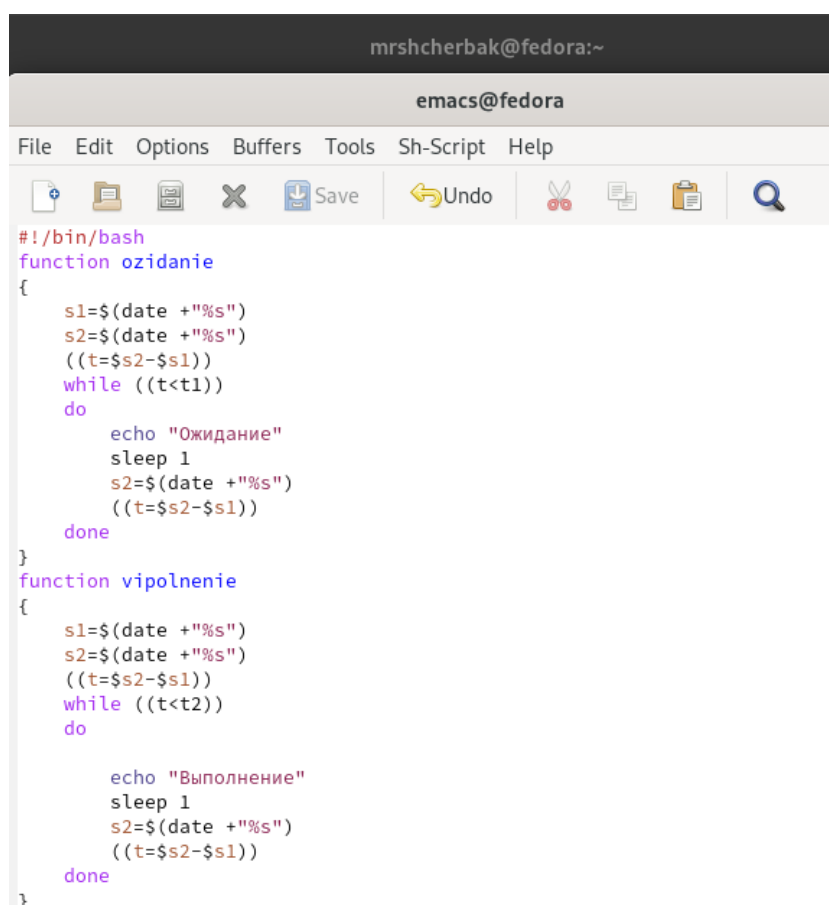
3. Используя встроенную переменную `$RANDOM`, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что `$RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

2 Теоретическое введение:

Выполнение условного оператора `if` сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `if`. Затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), то будет выполнена последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `then`. Фраза `elif` проверяется в том случае, когда предыдущая проверка была ложной. Строка, содержащая служебное слово `else`, является необязательной. Если она присутствует, то последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `else`, будет выполнена только при условии, что последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `if` или `elif`, возвращает ненулевой код завершения (ложь).

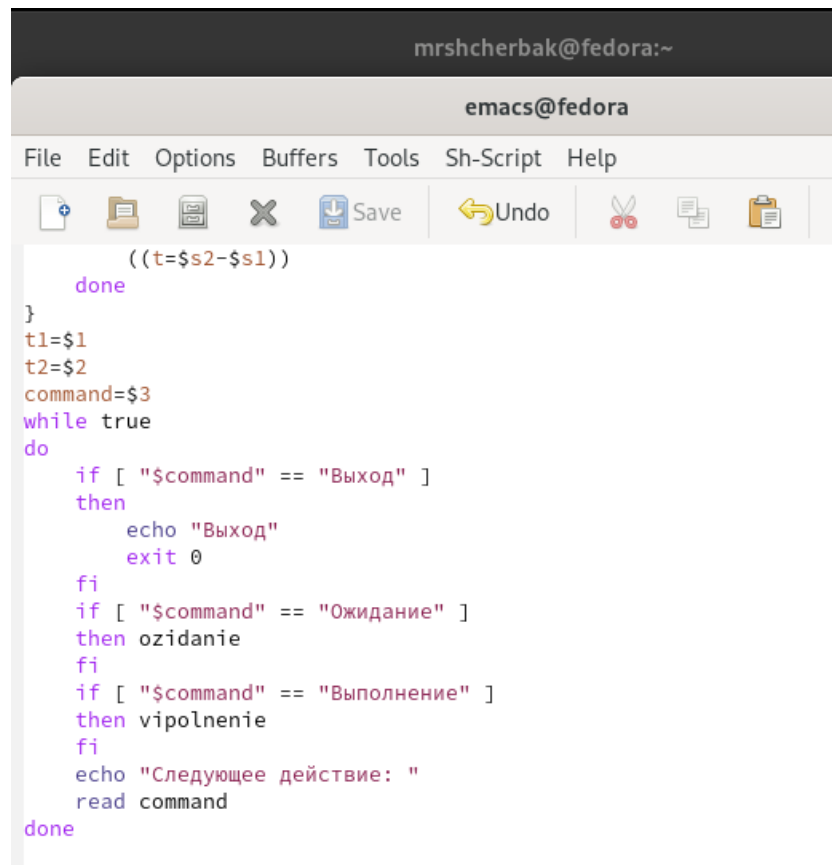
3 Выполнение лабораторной работы:

1. **Первое задание.** Для выполнения первого задания я создала файл prog11.sh, в котором писала скрипт, открыла текстовый редактор emacs. Также дала созданному файлу право доступа на выполнение (+x). Написала командный файл, реализующий упрощённый механизм семафоров. (Рис. 3.1 - Рис. 3.3).



```
#!/bin/bash
function ozidanie
{
    s1=$(date +%s")
    s2=$(date +%s")
    ((t=s2-s1))
    while ((t<t1))
    do
        echo "Ожидание"
        sleep 1
        s2=$(date +%s")
        ((t=s2-s1))
    done
}
function vipolnenie
{
    s1=$(date +%s")
    s2=$(date +%s")
    ((t=s2-s1))
    while ((t<t2))
    do
        echo "Выполнение"
        sleep 1
        s2=$(date +%s")
        ((t=s2-s1))
    done
}
```

Рис. 3.1: Скрипт 1го задания



```
mrshcherbak@fedora:~  
emacs@fedora  
File Edit Options Buffers Tools Sh-Script Help  
Save Undo  
((t=$s2-$s1))  
done  
}  
t1=$1  
t2=$2  
command=$3  
while true  
do  
  if [ "$command" == "Выход" ]  
  then  
    echo "Выход"  
    exit 0  
  fi  
  if [ "$command" == "Ожидание" ]  
  then ozidanie  
  fi  
  if [ "$command" == "Выполнение" ]  
  then vipolnenie  
  fi  
  echo "Следующее действие: "  
  read command  
done
```

Рис. 3.2: Скрипт 1го задания (продолжение)

```
mrshcherbak@fedora:~  
[mrshcherbak@fedora ~]$ ./prog11.sh 7 3 Ожидание > /dev/tty2 &  
[13] 8330  
  
[12]+ Остановлен ./prog11.sh 7 3 Ожидание > /dev/tty2  
[mrshcherbak@fedora ~]$ ./prog11.sh 7 3 Ожидание > /home/mrshcherbak/  
bash: /home/mrshcherbak/: Это каталог  
[mrshcherbak@fedora ~]$ ./prog11.sh 7 3 Ожидание /home/mrshcherbak/  
Ожидание  
Ожидание  
Ожидание  
Ожидание  
Ожидание  
Ожидание  
Следующее действие:  
Выполнение  
Выполнение  
Выполнение  
Выполнение  
Следующее действие:  
Выход  
Выход  
  
[13]+ Остановлен ./prog11.sh 7 3 Ожидание > /dev/tty2  
[mrshcherbak@fedora ~]$ ./prog11.sh 2 6 Ожидание > /dev/tty2 &  
[14] 8422  
[mrshcherbak@fedora ~]$ ./prog11.sh 2 6 Выполнение > /dev/tty2 &  
[15] 8443  
  
[14]+ Остановлен ./prog11.sh 2 6 Ожидание > /dev/tty2  
[mrshcherbak@fedora ~]$
```

Рис. 3.3: Выполнение

Делаем вывод, что скрипт работает корректно.

2. **Второе задание.** Создала файл prog12.sh, в котором писала второй скрипт, и открыла его в редакторе emacs.

Предоставила право доступа на выполнение файлу prog12.sh.

Командный файл получает в виде аргумента командной строки название команды и в виде результата выдаёт справку об этой команде/сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1. (Рис. 3.4 - Рис. 3.8).

```
command=""
while getopts :n: opt
do
    case $opt in
        n) command="$OPTARG";;
    esac
done
if test -f "/usr/share/man/man1/$command.1.gz"
then less /usr/share/man/man1/$command.1.gz
else
    echo "Нет такой команды"
fi
```

U:--- prog12.sh All L13 (Shell-script[sh])

Рис. 3.4: Скрипт 2го задания

```
[mrshcherbak@fedora ~]$ cd /usr/share/man/man1
[mrshcherbak@fedora man1]$ ls
.:.1.gz
'[,.1.gz'
a2ping.1.gz
ab.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
abrt-action-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
abrt-auto-reporting.1.gz
abrt-bodhi.1.gz
mkfifo.1.gz
mkfontdir.1.gz
mkfontscale.1.gz
mkhybrid.1.gz
mkindex.1.gz
mkisofs.1.gz
mkjobtexmf.1.gz
mkmanifest.1.gz
mkmapfile.1.gz
mknod.1.gz
mkocp.1.gz
mkofm.1.gz
mkpasswd.1.gz
mksquashfs.1.gz
mktemp.1.gz
mktexfmt.1.gz
mktexlsr.1.gz
mktexmf.1.gz
mktexpk.1.gz
mktextfm.1.gz
mlabel.1.gz
mm2gv.1.gz
mmafm.1.gz
mmcli.1.gz
mmd.1.gz
mmdblookup.1.gz
mmount.1.gz
mmove.1.gz
```

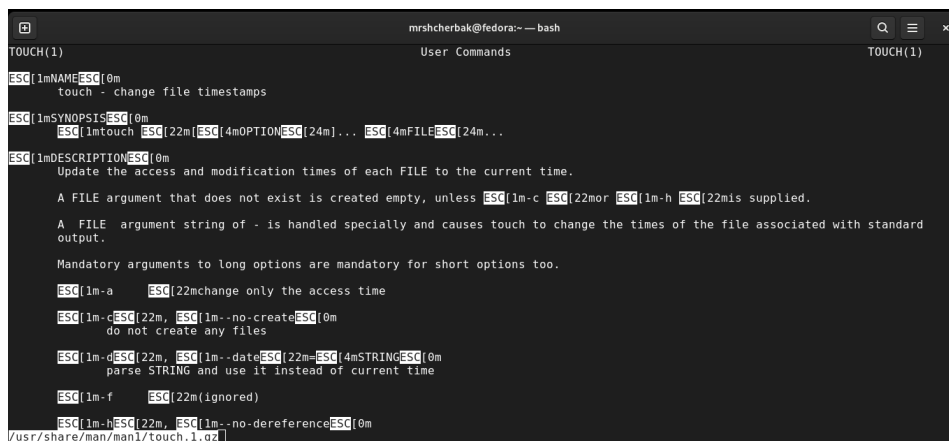
Рис. 3.5: Изучила содержимое каталога /usr/share/man/man1.


```

[mrshcherbak@fedora man1]$ cd ~
[mrshcherbak@fedora ~]$ touch prog12.sh
[mrshcherbak@fedora ~]$ chmod +x prog12.sh
[mrshcherbak@fedora ~]$ emacs &
[16] 8681
[mrshcherbak@fedora ~]$ ./prog12.sh -n touch
[mrshcherbak@fedora ~]$ ./prog12.sh -n kill
[mrshcherbak@fedora ~]$ ./prog12.sh -n du
[mrshcherbak@fedora ~]$ ./prog12.sh -n rm
\[mrshcherbak@fedora ~]$ ./prog12.sh -n shshgs
Нет такой команды
[mrshcherbak@fedora ~]$ █

```

Рис. 3.6: Выполнение



```

mrshcherbak@fedora:~ -- bash
TOUCH(1)
User Commands
TOUCH(1)

ESC[1mNAMEESC[0m
touch - change file timestamps

ESC[1mSYNOPSISESC[0m
ESC[1mtouch ESC[22m[ESC[4mOPTIONESC[24m]... ESC[4mFILEESC[24m]...

ESC[1mDESCRIPTIONESC[0m
Update the access and modification times of each FILE to the current time.

A FILE argument that does not exist is created empty, unless ESC[1m-c ESC[22mor ESC[1m-h ESC[22mis supplied.

A FILE argument string of - is handled specially and causes touch to change the times of the file associated with standard output.

Mandatory arguments to long options are mandatory for short options too.

ESC[1m-a ESC[22mchange only the access time

ESC[1m-cESC[22m, ESC[1m--no-createESC[0m
do not create any files

ESC[1m-dESC[22m, ESC[1m--dateESC[22m=ESC[4mSTRINGESC[0m
parse STRING and use it instead of current time

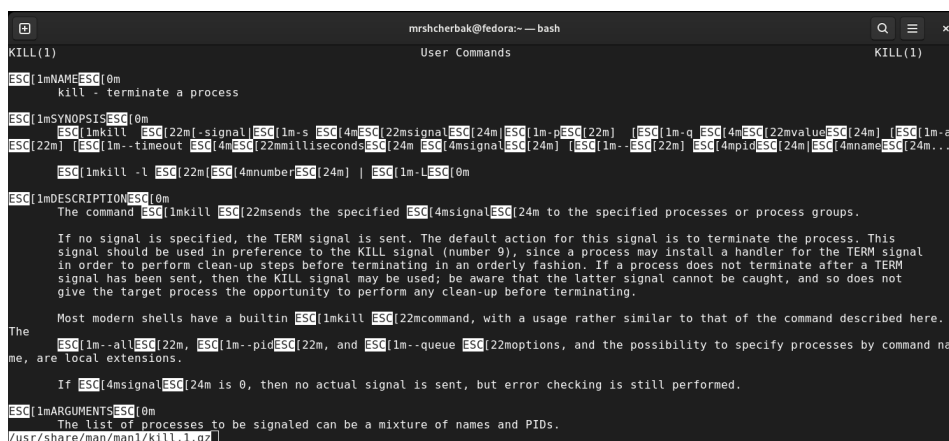
ESC[1m-f ESC[22m(ignore)

ESC[1m-hESC[22m, ESC[1m--no-dereferenceESC[0m

/usr/share/man/man1/touch.1.gz

```

Рис. 3.7: Проверка работы скрипта



```

mrshcherbak@fedora:~ -- bash
KILL(1)
User Commands
KILL(1)

ESC[1mNAMEESC[0m
kill - terminate a process

ESC[1mSYNOPSISESC[0m
ESC[1mkill ESC[22m[-signalESC[1m-s ESC[4mESC[22msignalESC[24m] ESC[1m-pESC[22m] [ESC[1m-q ESC[4mESC[22mvalueESC[24m] [ESC[1m-a
ESC[22m] [ESC[1m--timeout ESC[4mESC[22mmillisecondESC[24m] ESC[4msignalESC[24m] [ESC[1m--ESC[22m] ESC[4mpidESC[24m] ESC[4mnameESC[24m]...
ESC[1mkill -l ESC[22m[ESC[4mnumberESC[24m] | ESC[1m-lESC[0m

ESC[1mDESCRIPTIONESC[0m
The command ESC[1mkill ESC[22msends the specified ESC[4msignalESC[24m to the specified processes or process groups.

If no signal is specified, the TERM signal is sent. The default action for this signal is to terminate the process. This signal should be used in preference to the KILL signal (number 9), since a process may install a handler for the TERM signal in order to perform clean-up steps before terminating in an orderly fashion. If a process does not terminate after a TERM signal has been sent, then the KILL signal may be used; be aware that the latter signal cannot be caught, and so does not give the target process the opportunity to perform any clean-up before terminating.

Most modern shells have a builtin ESC[1mkill ESC[22mcommand, with a usage rather similar to that of the command described here.

The ESC[1m--allESC[22m, ESC[1m--pidESC[22m, and ESC[1m--queue ESC[22moptions, and the possibility to specify processes by command name, are local extensions.

If ESC[4msignalESC[24m is 0, then no actual signal is sent, but error checking is still performed.

ESC[1mARGUMENTSESC[0m
The list of processes to be signaled can be a mixture of names and PIDs.

/usr/share/man/man1/kill.1.gz

```

Рис. 3.8: Справка по заданной команде

Таким образом, мы видим, что задание выполнено успешно.

3. **Третье задание.** Создала файл prog13.sh, в котором писала третий скрипт, и открыла его в редакторе emacs. Предоставила право доступа на выполнение файлу prog13.sh.

Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. (Рис. 3.9 - Рис. 3.10).

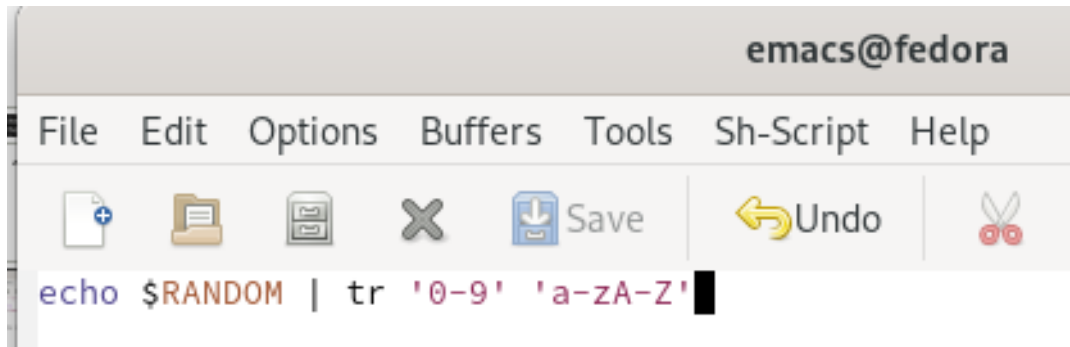


Рис. 3.9: Скрипт 3го задания

```
[mrshcherbak@fedora ~]$ touch prog13.sh
[mrshcherbak@fedora ~]$ chmod +x prog13.sh
[mrshcherbak@fedora ~]$ emacs &
[17] 9040
[mrshcherbak@fedora ~]$ ./prog13.sh
ciahb
[mrshcherbak@fedora ~]$ ./prog13.sh
dabhh
[mrshcherbak@fedora ~]$ ./prog13.sh
chghj
[mrshcherbak@fedora ~]$ ./prog13.sh
bbdcc
[mrshcherbak@fedora ~]$ ./prog13.sh
bida
[mrshcherbak@fedora ~]$ ./prog13.sh
cddih
[mrshcherbak@fedora ~]$ ./prog13.sh
gacd
[mrshcherbak@fedora ~]$
```

Рис. 3.10: Выполнение

Скрипт работает корректно.

4 *Контрольные вопросы:*

1). while [\$1 != "exit"]

В данной строчке допущены следующие ошибки:

не хватает пробелов после первой скобки [и перед второй скобкой]

выражение \$1 необходимо взять в "", потому что эта переменная может содержать пробелы.

Таким образом, правильный вариант должен выглядеть так: while ["\$1"!= "exit"]

2). Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами:

Первый: VAR1="Hello,

"VAR2=" World"

VAR3="VAR1VAR2"

echo "\$VAR3"

Результат: Hello, World

Второй: VAR1="Hello,"

VAR1+= " World"

echo "\$VAR1"

Результат: Hello, World

3). Команда seq в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT.

Параметры:

seq LAST: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение is не выдает.

seq FIRST LAST: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1, равным 1. Если LAST меньше FIRST, он не выдает никаких выходных данных.

seq FIRST INCREMENT LAST: когда заданы три аргумента, он генерирует числа от FIRST до LAST на шаге INCREMENT . Если LAST меньше, чем FIRST, он не производит вывод.

seq -f «FORMAT» FIRST INCREMENT LAST: эта команда используется для генерации последовательности в форматированном виде. FIRST и INCREMENT являются необязательными.

seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО: Эта команда используется для STRING для разделения чисел. По умолчанию это значение равно /n. FIRST и INCREMENT являются необязательными.

seq -w FIRST INCREMENT LAST: эта команда используется для выравнивания ширины путем заполнения начальными нулями. FIRST и INCREMENT являются необязательными.

4). Результатом данного выражения $\$(10/3)$ будет 3, потому что это целочисленное деление без остатка.

5). Отличия командной оболочки zsh от bash:

В zsh более быстрое автодополнение для cdc помощью Tab

В zsh существует калькулятор zcalc, способный выполнять вычисления внутри терминала

В zsh поддерживаются числа с плавающей запятой

В zsh поддерживаются структуры данных «хэш»

В zsh поддерживается раскрытие полного пути на основе неполных данных

В zsh поддерживается замена части пути

В zsh есть возможность отображать разделенный экран, такой же как разделенный экран vim

6). for((a=1; a<= LIMIT; a++)) синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().

7). Преимущества скриптового языка bash:

Один из самых распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS

Удобное перенаправление ввода/вывода

Большое количество команд для работы с файловыми системами Linux

Можно писать собственные скрипты, упрощающие работу в Linux

Недостатки скриптового языка bash:

Дополнительные библиотеки других языков позволяют выполнить больше действий

Bash не является языком общего назначения

Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта

Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий.

5 Выводы

Таким образом, в ходе ЛРН^{№12} я изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.